Programowanie w języku Java - Wprowadzenie

mgr inż. Maciej Lasota <m.lasota@tu.kielce.pl>

Version 1.0, 27-02-2017

Spis treści

Programowanie obiektowe (repetytorium, podstawy)

Programowanie obiektowe jest jednym z `paradygmatów programowania`, które opiera się o tworzenie aplikacji w taki sposób, aby jak najlepiej odzwierciedlać otaczającą nas rzeczywistość i aby model danych oddawał sposób postrzegania świata przez człowieka (czyli, że na przykład jabłko jest owocem, a nie kolorową kulą).

Za pierwszy prawdziwie obiektowy język programowania uważany jest **Simula 67**, który powstał w latach 60-tych ubiegłego stulecia. Język ten powstał podczas pracy nad symulacją statków. To w tym języku po raz pierwszy wprowadzono pojęcie *klasy* i *egzemplarza danej klasy*. Dzięki temu językowi możliwa była *tzw. symulacja*, czyli odwzorowanie obiektów świata rzeczywistego na obiekty używane w programie.

Idea programowania obiektowego została następnie dopracowana w języku **Smalltalk (1971)**, w którym obiekty mogą być tworzone i modyfikowane dynamicznie, tzn. w trakcie działania programu (w przeciwieństwie do statycznych programów).

Powstanie języka **C++ (1983)** przyczyniło się w sposób szczególny do rozpowszechnienia idei programowania obiektowego. Cechy obiektowości pojawiły się również w wielu innych językach programowania takich jak np. *Ada, Eiffel, Basic, Pascal, Lisp*.

Dzisiaj jednym z najpopularniejszych obiektowych języków programowania jest **Java**.

Przykłady innych obiektowych języków programowania: **Python, Perl, C#, Ruby, Ocaml, PHP5**.

Elementarnym oraz fundamentalnym pojęciem w programowaniu obiektowym jest **obiekt**.

Obiekt jest abstrakcyjnym bytem opisującym i odzwierciedlającym rzeczywistość. Każdy obiekt jako struktura składa się z **danych** *(właściwości, atrybutów, pól)* oraz **metod** *(zachowań, funkcji)* pozwalających operować na tych danych.

Właściwości obiektów:

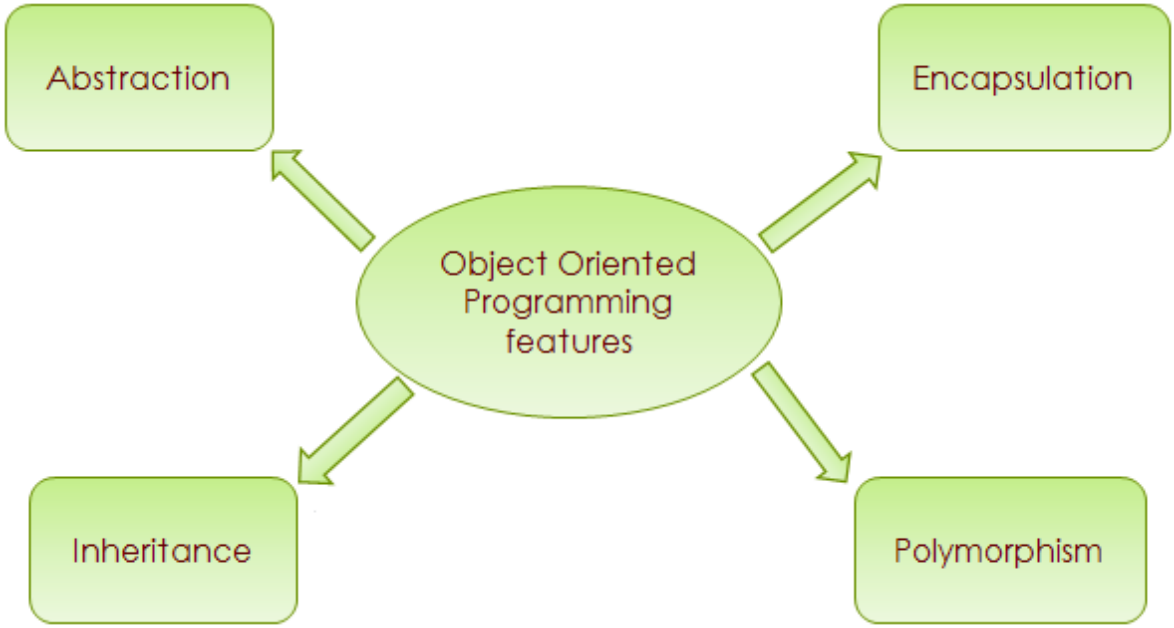
- **Zachowanie obiektu** – co można zrobić dzięki temu obiektowi i jakie metody (funkcje) można dla niego wywoływać.
- **Stan obiektu** – jak obiekt reaguje na działanie tych metod.
- **Tożsamość obiektu** – w jaki sposób można odróżnić ten obiekt od innych, posiadając to samo zachowanie i stan.

Do opisu i definiowania obiektów w programowaniu obiektowym wykorzystywane są klasy.

Klasa to kolejne podstawowe pojęcie programowania obiektowego, jest to pojęcie *abstrakcyjne* oznaczające własny *typ danych* częściowo lub całkowicie definiujące obiekty. Klasa jest najważniejszym z pojęć związanym z programowaniem zorientowanym obiektowo.

Klasa jest szablonem, czy też projektem na podstawie którego tworzone są obiekty, które posiadają pewne **cechy** i **funkcje**. Zatem klasa jest narzędziem, za pomocą którego tłumaczy się abstrakcję na typ użytkownika.

Wyróżniamy **cztery** podstawowe założenia *paradygmatu programowania obiektowego*:



Rysunek 1: Założenia programowania obiektowego

1. **Abstrakcja** *(ang. Abstraction)*

   ```
   Abstrakcją w programowaniu obiektowym nazywamy cechę, polegającą na tym, że pewne klasy są jedynie wzorem dla innych klas, które się z niej wywodzą, ale same nie są wykorzystywane do tworzenia obiektów. Abstrakcja ma za zadanie uprościć rozwiązanie problemu i zwiększyć jego ogólność.
   ```

2. **Hermetyzacja** *(ang. Encapsulation)*

   ```
   Hermetyzacja zwana inaczej enkapsulacją, jest kluczowym zagadnieniem programowania zorientowanego obiektowo. Polega ona na ukrywaniu implementacji przed użytkownikiem obiektu. Hermetyzacja zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób.
   ```

3. **Polimorfizm** *(ang. Polymorphism)*

```
Polimorfizm umożliwia dostosowanie działania obiektów do własnych oczekiwań poprzez
łączenie funkcjonalności zarówno dziedziczonej, jak i implementowanej samodzielnie.
Idea polimorfizmu bazuje na tym, że użytkownik obiektu nie wie i nie musi wiedzieć,
czy konkretne zachowanie wykorzystywanego obiektu zostało zrealizowane bezpośrednio
w tym obiekcie czy też w tym, po którym dziedziczy on swoje właściwości.
```

4. **Dziedziczenie** *(ang. Inheritance)*

```
Dziedziczenie umożliwia definicję i tworzenie nowych obiektów na podstawie już
istniejących obiektów bardziej ogólnych.
```

Język Java, główne koncepcje i założenia

Java jest *uniwersalnym językiem programowania* stworzonym przez grupę roboczą pod kierunkiem *Jamesa Goslinga* z firmy **Sun Microsystems** w roku 1995. Obecnie za rozwijanie języka Java oraz technologi z nią związanych odpowiada firma **Oracle** (po przejęciu firmy Sun Microsystems). Oprócz wersji języka Java tworzonej przez firmę Oracle (https://www.oracle.com/pl/java/index.html) istnieje też wersja o otwartym kodzie źródłowym **OpenJDK** (http://openjdk.java.net/).

Java Version/Code Name	Release Date	Important Features/Code Name
JDK Alpha and Beta	1995	Initial release
JDK1.0(OAK)	23rd Jab 1996	Initial release
JDK1.1	19th Feb 1997	Reflection ,JDBC, Inner Classes, MI
J2SE1.2(Playground)	8th Dec 1998	Collection,JIT,String memory map
J2SE 1.3(Kestrel)	8th May 2000	Java Sound,Java Indexing,JNDI
J2SE 1.4(Merlin)	6th Feb 2002	Assert,regex,exception chaining
J2SE 5.0(Tiger)	30th Sept 2004	Generic,autoboxing,enums,varargs,for each,static import
Java SE 6.0(Mustang)	11th Dec 2006	JDBC4.0,Java compiler API Annotations
Java SE7.0(Dolphin)	28th July 2011	String in switch case, resource management in exception,catching multiple exception
Java SE8.0	18th March 2014	Lambad expression,Annotation on Java type,Data and Time API

Rysunek 2: Historia wersji języka Java

Składniowe *podobieństwo* do **C/C++** czyni ten język łatwy do opanowania przez programistów znających te języki. Jednocześnie Java ma ambicje udoskonalania swoich wzorców. Programista Javy w zasadzie nie musi martwić się zarządzaniem pamięcią.

W Javie funkcjonuje **automatyczne odśmiecanie** – *"garbage collection"*, polegające na automatycznym usuwaniu przydzielonych wcześniej, a nieużywanych obszarów pamięci.

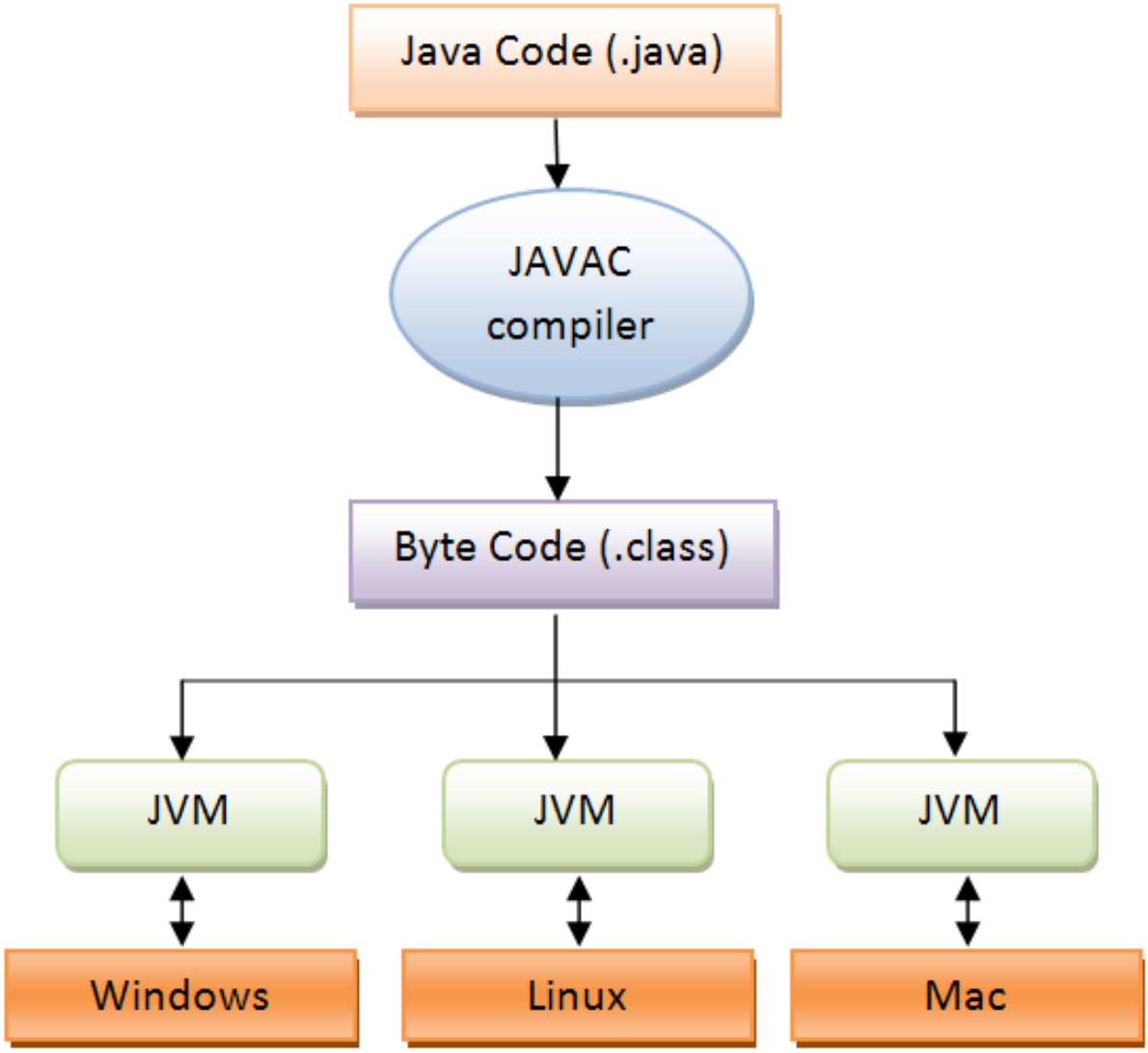
Java nie dopuszcza **arytmetyki wskaźnikowej**, która pozwala na odwoływanie się do dowolnych obszarów pamięci i jest częstą przyczyną błędów.

Ścisła kontrola zgodności typów na etapie kompilacji pozwala unikać prostych błędów. Co więcej to tzw. *statyczne typowanie* umożliwia **zintegrowanym środowiskom programowania** *(IDE)* przebogate wspieranie programisty przy pisaniu i testowaniu programu (automatyczne dopisywanie kodu i poprawianie błędów). **Konwersje** *(rzutowanie)* typów przeprowadzane w fazie wykonania są bezpieczne, bowiem nigdy nie może powstać sytuacja przekształcenia danych do niewłaściwego dla nich typu.

Wymuszana przez kompilator obsługa niektórych wyjątków (inaczej mówiąc - błędów) czyni programowanie w Javie jeszcze bardziej **bezpiecznym** i **niezawodnym**, a wbudowane w język podstawowe elementy **współbieżności** umożliwiają łatwe tworzenie i synchronizowanie równolegle działających wątków (czyli równolegle wykonywanych fragmentów tego samego programu).

Java jest językiem **interpretowanym**, co umożliwia wykonywanie *"binarnych"* kodów Javy bez rekompilacji praktycznie na wszystkich platformach systemowych.

Kod źródłowy *(pliki z rozszerzeniem ".java")* jest kompilowany przez kompilator Javy *(program javac)* do **kodu bajtowego** *(Byte Code, B-kodu, pliki z rozszerzeniem ".class")* , ten ostatni zaś jest interpretowany przez tzw. `wirtualną maszynę Javy` – **JVM** (jest to program wraz odpowiednimi dynamicznymi bibliotekami), zainstalowaną na danej platformie systemowej.



Rysunek 3: Proces kompilacji i uruchamiania kodu języka Java

Słownik pojęć i definicji, technologie, narzędzia oraz środowiska

Słownik pojęć i definicji:

- **JVM** *(Java Virtual Machine)* – maszyna wirtualna oraz środowisko uruchomieniowe zdolne do wykonywania kodu bajtowego Javy (aplikacji Java). Wirtualne maszyny Java są dostępne dla wszystkich najpopularniejszych platform sprzętowych i programowych.
- **GC** *(Garbage Collector)* – część składowa (proces) JVM do automatycznego zarządzania dynamicznie przydzieloną pamięcią.
- **JRE** *(Java Runtime Environment)* – środowisko zawierające JVM oraz standardowe biblioteki języka Java służące do uruchamiania aplikacji napisanych w języku Java.
- **JDK** *(Java Development Kit)* – środowisko zawierające JRE wraz z kompilatorem, debugerem, narzędziami tworzenia dokumentacji i innymi narzędziami pomocniczymi służące do tworzenia oraz uruchamiania aplikacji napisanych w języku Java.
- **SDK** *(Software Development Kit)* – zawiera JDK oraz kompletny zestaw pomocy.

- **API** *(Application Programing Interfaces)* - specyfikacja metod i interfejsów.
- **IDE** *(Integrated Development Environment)* - zintegrowane środowisko programistyczne.

Technologie (wersje Java):

- **Java Standard Edition** *(Java SE, oficjalnie Java Platform™, Standard Edition, dawniej J2SE do wersji 5.0)* – podstawowa wersja Java przeznaczoną głównie do standardowych zastosowań dla komputerów personalnych oraz serwerów, również połączonych w sieci. Występuje w trzech rodzajach Java SE JDK, Java SE JRE oraz Java SE JRE Server. Aktualna wersja: Java SE 8 http://www.oracle.com/technetwork/java/javase/overview/index.html
- **Java Enterprise Edition** *(Java EE)* – rozszerzona wersja Java przeznaczona do tworzenia rozbudowanych i zaawansowanych aplikacji biznesowych, w szczególności dla dużych firm oraz korporacji. Występuje w dwóch rodzajach Java EE SDK oraz Java EE SDK Web Profile. Aktualna wersja: Java EE 7 http://www.oracle.com/technetwork/java/javaee/overview/index.html
- **Java Micro Edition** *(Java ME)* - wersja Java przeznaczona do programowania urządzeń elektronicznych, takich jak telefony komórkowe, telewizja, procesory w samochodach czy urządzeniach gospodarstwa domowego. Występuje w kilku wersjach Java ME Embedded, Java ME SDK, Java ME Embedded Client, Java for Mobile, Java TV. Aktualna wersja: Java ME Embedded 8.3, Java ME SDK 8.3 http://www.oracle.com/technetwork/java/embedded/javame/index.html

Narzędzia:

- **javac** - kompilator języka Java (source → bytecode)
- **java** - środowisko JVM do uruchamiania skompilowanego kodu (z konsolą)
- **javaw** - środowisko JVM do uruchamiania skompilowanego kodu (bez konsoli)
- **javadoc** - generator dokumentacji (source → apidoc)
- **appletviewer** - przeglądarka apletów
- **jar** - program do zarządzania archiwami jar
- **jdb** - debugger

Zintegrowane środowiska programistyczne:

- **Eclipse IDE** - https://eclipse.org/
- **NetBeans IDE** - https://netbeans.org/
- **Intellij IDEA** - https://www.jetbrains.com/idea/

Kompilacja kodu, uruchamianie aplikacji

Kompilacja (generowany jest plik NazwaKlasy.class)

```
javac NazwaKlasy.java
```

Uruchamianie (skompilowanego kodu NazwaKlasy.class)

```
java NazwaKlasy
javaw NazwaKlasy
```

```
appletviewer Applet.html
```

gdzie Applet.html:

```
<html>
  <applet code=NazwaKlasy.class width=320 height=240>
  </applet>
</html>
```

Archiwizacja

```
jar -cf NazwaArchiwum.jar *.class
```

Bibliografia

- **Bruce Eckels**, *"Thinking in Java. Edycja polska. Wydanie IV"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Podstawy. Wydanie IX"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Techniki zaawansowane. Wydanie IX"*, wydawnictwo Helion.
- **Krzysztof Barteczko**, *"Podstawy programowania w języku Java, PJWSTK"*, http://edu.pjwstk.edu.pl/wyklady/ppj/scb/
- **Konrad Kurczyna**, *"Laboratorium Java"*, Politechnika Świętokrzyska w Kielcach.
- **Dariusz Wardowski**, *"Języki i Paradygmaty Programowania"*, Katedra Analizy Nieliniowej, WMiI UŁ, http://math.uni.lodz.pl/~wardd/images/dydaktyka/paradygmaty%20programowania/
- **Oracle Centrum Pomocy Java**, https://www.java.com/pl/download/help/